



Rewriting Techniques in the Constraint Solver

Jonathan Millen

*The MITRE Corporation
Bedford, MA, USA*

Abstract

The constraint solver is a symbolic cryptographic protocol security analysis tool that is based on a unique term rewriting approach. Several of the design characteristics of this tool, and the reasons for them, are discussed and placed in perspective with other approaches. The constraint solver uses a free message algebra and a bounded-process network model with a Dolev-Yao attacker. These choices yield simplicity and decidability.

Keywords: cryptographic protocol, security analysis, term rewriting, constraint solver

1 Introduction

Symbolic analysis of security protocols originated with the Dolev-Yao paper [21], which was first published in 1981 as a Stanford report. Their idea was to represent cryptography abstractly in an algebraic context. A public-key cryptosystem had a pair of operators, E_a and D_a , for encryption and decryption, for each user a that might participate in the protocol. The term $E_a(M)$ represented the result of encrypting M with a 's public key, and $D_a(M)$ was the decryption of M using a 's corresponding private key. There were two term rewriting rules, which applied to a composition of operators: $D_a E_a$ reduces to the identity operator, and so does $E_a D_a$. It was assumed that anyone could use the encryption operator for any user, but only a could apply the decryption operator D_a . Besides the encryption and decryption operators, there was also a “name stamp” operator and its inverse that appends or removes a user name after a message.

By itself, this is only a model of a message algebra. There was also a model of protocols and a network environment. The only protocols that were considered were a simple kind that were called “Ping-Pong” protocols in a subsequent paper [20]. These are two-party protocols in which the first message has the form $\alpha_1(M)$, where M is a secret, and α_1 is a composition of operators. Subsequent responses

and replies could only apply more operators to the received term and send it out again, so that all available messages have the same form $\alpha(M)$.

There have been many extensions and modifications over the years to the message algebra and protocol specification style, but the Dolev-Yao characterization of the network environment with its “saboteur” has remained essentially the same in symbolic protocol analysis. The saboteur is both a user z and an active eavesdropper who can modify any message in transit. Messages are modified by applying a string of operators to any known message, and the result may be delivered to any user. The saboteur is limited to D_z for decryption. The protocol is insecure if the saboteur can expose the original secret M , by reducing the operator sequence α on some message to the identity.

This characterization has a subtlety that is sometimes neglected in attempts to develop new analysis techniques. Note that messages modified by the saboteur can be fed back to an honest user to obtain the transformational effect of whatever operators that user applies at any stage of the protocol. And if the user has reached stage n in its role, the saboteur can still get a transformation from stage $m < n$ by starting up a new concurrent session of that (or another) user. A successful attack might require many sessions of the same user to be started up. We don’t have to know how they can be forced by the saboteur, just that they are possible.

Terminology has changed over the years. Many synonyms for “saboteur” have been used, such as penetrator, attacker, intruder, and spy. A user is referred to as a principal or agent, and the concept of principal has been extended to include any user agent that controls some secret key.

2 Motivation

Before going into more detail on the constraint solver and the role of term rewriting, it might be best to say a little about why protocol analysis was seen as desirable, and what other approaches are being used.

The need for some kind of careful analysis of cryptographic protocols was first made known publicly by the Denning-Sacco paper [17], which demonstrated a replay attack on the symmetric-key Needham-Schroeder protocol [40]. A replay attack is one in which the attacker records the message exchanges from a normal run of a protocol, and then actively inserts those messages, or parts of them, into a subsequent run, to compromise the protocol. In the Denning-Sacco example, it was assumed that a session key used in the earlier normal run had eventually been revealed somehow. Then the replay attack caused that same key to be accepted in the later run instead of a new key.

The concept of a replay attack was already known in some circles, at the time, along with other types having colorful names such as “masquerading” and “man-in-the-middle”. An algebraic approach supersedes these rubrics, just as first-order logic supersedes syllogisms, although they can still help to support informal, intuitive analysis.

My own view of the broader picture of cryptographic protocol analysis is shown

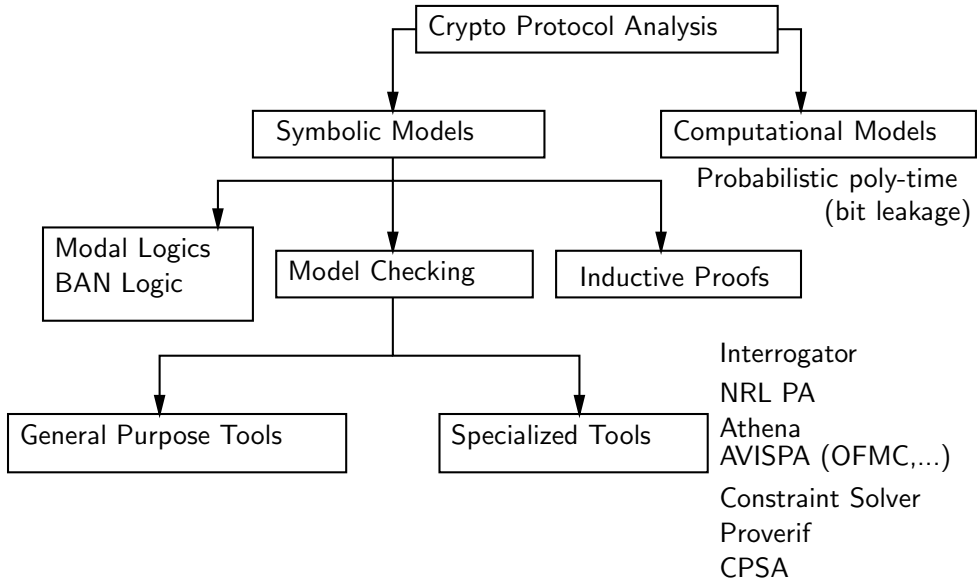


Fig. 1. Protocol Analysis Methods

in Figure 1. This picture categorizes analysis approaches according to the tools and techniques used to perform the analysis. At the top level, symbolic methods, sometimes called formal methods, are distinguished from methods employing computational models. Symbolic methods had their origin in the Dolev-Yao model; computational models arise from methods used in the analysis of cryptographic algorithms.

The main difference between the two is that computational models can recognize leakage of information from an encrypted message that is less than the entire message, while in symbolic approaches a message is either entirely compromised or safe. An example of the kind of problem that is not detectable symbolically, but which should be found computationally, is Bleichenbacher’s “million-message” attack on a public key standard protocol [8]. The ciphertext is revealed by combining fractional bits of information from many runs. Computational approaches treat key guessing as a probabilistic exponential time activity, as opposed to the symbolic approach of just considering key guessing as impossible. One formulation of the computational approach is given in [28].

In the last few years, considerable effort has been put into reconciling symbolic and computational approaches, beginning with the Abadi and Rogaway paper [1]. The idea is to prove that symbolic conclusions are valid if the real cryptographic operations satisfy sufficiently strong properties. A fine-grained categorization of relevant security definitions is given in [5], including such properties as indistinguishability under chosen-ciphertext attacks. A library of operations proved to have strong enough properties has been proposed [3]. As pointed out by Blanchet [7], however, using strongly secure libraries is not as easy as one might hope, because the operations in them do not exactly match the ones axiomatized in most symbolic models, and additional assumptions and restrictions might apply to the

protocols and the environment in which they are used.

Symbolic protocol analysis has been conducted using several different tools and techniques, as shown in Figure 1. One elegant but limited approach originated with BAN logic, a specialized modal logic that required a protocol to be idealized into logical statements [10]. The idealization step had underlying assumptions that could not be adequately systematized, so despite several papers proposing different versions of it, it is now used primarily as an aid to intuition.

The goal of proving protocols correct was carried on in the second thread, using various types of state-transition specifications of protocols as the object of inductive proofs that data was shared properly among authorized participants. Such proofs must take into account attacker actions and multiple sessions. Both hand proofs and software tool-supported proofs were demonstrated; see, for example, [9,41,44,46,13]. It turns out that a rigorous security proof for a protocol is a long, arduous, error-prone procedure, with a similar flavor to a program proof. The first tool-supported proof that was well enough explained to encourage imitation using other provers was by Paulson [41].

A few researchers hit upon the idea of using a model checker to explore the state space of a cryptographic protocol. Roscoe might have been the first [42]. Using a model checker generally means making some approximation or imposing some theoretically severe limitation on the protocol and environment model. But this disadvantage is counterbalanced by the ease of specifying the protocol and the ability to explore modifications in the protocol and environmental assumptions quickly and easily. And over the years, model checkers and analysis techniques have improved. Model checking runs took an hour or so at first, which was still better than the days needed to update an inductive proof after a modification, but now a few seconds will usually do.

Model checking in the general sense was being done from quite early on, in the sense that there were specialized tools for exploring the protocol state space [37]. The real boost for protocol analysis came from the use of general-purpose model checkers built for other applications. Some early examples of this approach are [29,39,16]. This way, many groups of researchers could use well-supported, or at least locally supported, tools that they already had, rather than having to import experimental software with which they were unfamiliar. Also, the limitations imposed by practical model checking were clear and easily understood, while specially designed analyzers were too complex to understand exactly what attacks they might miss. In fact, the use of general-purpose model checkers had the side effect of raising the standards for symbolic state-exploration protocol analysis, so that newer specialized tools and newer versions of older tools are much improved in their conceptual clarity.

The specialized tools listed in Figure 1 are just a small sample that includes a few old and new tools. They are: the Interrogator [37], the Naval Research Laboratory Protocol Analyzer [34], Athena [45], the “On-the-fly Model Checker” [4], Proverif [6], and the Cryptographic Protocol Shape Analyzer [19].

3 Protocol Analysis With The Constraint Solver

One of the protocol analysis methods that uses a kind of rewriting approach is the constraint solver, which I developed with Vitaly Shmatikov’s help [38], making use of observations and suggestions from others at the SRI Computer Science Lab, particularly Harald Ruess and Bruno Dutertre. The constraint solver adopts the strand space model [46] as a way to specify protocols, but does not use the authentication test analysis approach [18]

The constraint solver has two phases. The first phase uses the protocol specification to generate a large number of systems of derivation constraints. Each constraint system corresponds to a potential message event sequence.

Then, in the second phase, it tries to solve the constraint systems. A solved system yields a realizable event sequence that can be interpreted as an attack on the protocol. A constraint system is solved, or found to be unsolvable, using a rewriting procedure.

A trivial example will help to explain how it works. Consider the following (laughably insecure) protocol:

$$\begin{aligned} A &\rightarrow B : [A, M_{\text{pk}(B)}] \\ B &\rightarrow A : M_{\text{pk}(A)} \end{aligned}$$

The notation $X_{\text{pk}(Y)}$ represents the encryption of X with the public key of Y , and $[X, Y]$ is the concatenation of X and Y .

3.1 Generation Phase

As in many protocol analysis approaches, the “Alice and Bob” notation must be turned into a process view where the roles of the initiator and responder are separated. We get the two roles:

| <i>Initiator</i> (A, B, M) | <i>Responder</i> (B, A, M) |
|--------------------------------|---------------------------------|
| send $[A, M_{\text{pk}(B)}]$ | receive $[A, M_{\text{pk}(B)}]$ |
| receive $M_{\text{pk}(A)}$ | send $M_{\text{pk}(A)}$ |

Note that identifiers A, B, M occurring as parameters of a role may be instantiated differently in another role or in another instance of the same role.

At this point we must face the main limitation of the constraint solver, and that is its *bounded-process* limitation. Recall that, in the Dolev-Yao environment model, any number of instances of each role may be started up. The bounded-process limitation requires us to state in advance how many instances of each role we will consider during the analysis. Imposing this bound makes the secrecy problem decidable. This was observed by Huima [27] and proved with an NP-completeness result by Rusinowitch and Turuani [43].

Although the bounded-process limitation may cause us to miss some attack on the protocol, it has the useful advantage that the security problem (more specifically, the message secrecy problem) is decidable under this assumption, whereas it is undecidable otherwise (if no other limitation is imposed). Undecidability of

the security of cryptographic protocols has been established many times; see, for example, [26,22]. The security of ping-pong protocols is decidable in polynomial time, but adding a second encrypted field, for example, pushes the problem over the edge into undecidability [23].

Suppose, for now, that we allow only one instance of each role, and omit the receive event for A , which does not affect the secrecy goal. We generate all possible message event sequences consistent with the order of messages in each role. In this example there are only three possibilities:

| | | |
|--------------------------------------|--------------------------------------|--------------------------------------|
| A sends $[A, M_{\text{pk}(B)}]$ | B' recvs $[A', M_{\text{pk}(B')}]$ | B' recvs $[A', M_{\text{pk}(B')}]$ |
| B' recvs $[A', M_{\text{pk}(B')}]$ | A sends $[A, M_{\text{pk}(B)}]$ | B' sends $M'_{\text{pk}(A')}$ |
| B' sends $M'_{\text{pk}(A')}$ | B' sends $M'_{\text{pk}(A')}$ | A sends $[A, M_{\text{pk}(B)}]$ |

Note that parameters have been renamed in the responder role since the responder may disagree with the initiator on the identity of the initiator or other parameters. The scope of a parameter variable is now the whole event sequence.

The difference between a potential event sequence and a *realizable* event sequence is that each received message must be derivable by the attacker from available data. The attacker can use messages that were previously sent, subterms of them obtained by decomposing them, and whatever data the attacker is assumed to know in advance, such as public keys, the attacker's own private key, protocol-related constants, and names of principals.

If there are more role instances and they are longer, the number of possible event sequences can get large. For two role instances of lengths m and n , the number of potential mergings is the number of binomial combinations $C(m+n, n)$. However, there are some optimizations that can reduce the number that need to be considered.

What matters is which messages have been sent when a received message must be derived. Consequently, if two consecutive send events occur, their order does not matter. Also, a send event can occur as soon as the prior receive event in the same role, and delaying it results only in a sequence that is less likely to be realizable. Putting these two observations together, the number of potential sequences that need to be considered is found by measuring the length of a role by the number of received messages only. In the example, we can get by with just one sequence (the first).

In other examples, the number of potential event sequences is still so large that further optimizations are desirable. Corin and Etalle [15] took advantage of the fact that many sequences share a common prefix. If a partial constraint system is generated for a prefix, and found to be unsolvable, then one can eliminate all the sequences with that prefix. This seems to have a substantial benefit. Another optimization by Basin, et al. [4] called *constraint differentiation* yields a further improvement.

Each event sequence results in a system of derivation constraints, one for each receive event. A derivation constraint states that a received message term is derivable from a set of available terms. The single receive event in the example sequence

generates the constraint

$$[A, M_{\text{pk}(B)}] \vdash [A', M'_{\text{pk}(B')}].$$

The meaning of “derivability” will be discussed momentarily, but it should be obvious that this constraint is solved by the unifying substitution $A = A'$, $B = B'$, and $M = M'$. In general, “solvability” means that there exists a global substitution on the parameters of all role instances such that the derivability constraints are proved to be satisfied by applying the available rules. Thus, solvability of the constraint system is required for realizability of the event sequence.

The solution above does not mean that the protocol is insecure, merely that there exists a realizable instance of a normal run of the protocol. To test for a compromise of the message M , we can add a “listener” role to the protocol with the single event $\text{recv}(M)$. Appending this event to the end of the event sequence (the best and sufficient place to put it), we get a second constraint:

$$[A, M_{\text{pk}(B)}], M'_{\text{pk}(A')} \vdash M.$$

We do not want the solution that sets $M = A$, because M is a secret and A is not. We can enforce this exclusion by introducing symbolic constants a and m such that $A = a$ and $M = m$. This works because the constraint solver’s message term algebra is *free*. In a free algebra, constants with different names are not equal, so this prevents the uninteresting solution $A = M$.

The other incorrect solution that we must guard against arises from the Dolev-Yao assumption that the attacker is a principal. If the attacker’s name is e , and we use the substitution $B = e$, the second constraint becomes

$$[a, m_{\text{pk}(e)}], m_{\text{pk}(a)} \vdash m.$$

A rule to derive m from $m_{\text{pk}(e)}$ is always assumed, since e is the attacker. So this constraint is solved. But the solution is trivial, because it says only that the initiator has deliberately chosen to share m with e . This is not a compromise.

We can avoid this spurious solution by introducing another constant b to instantiate B , making B distinct from A and from the attacker e . In effect, we have had to pin down the secret m and the two principals who are expected to share it, a and b . And we did this by instantiating parameters of only that role instance that created the secret m . (We might also want to consider a scenario with $A = B = a$, since that is not excluded by the protocol specification.)

With the starting substitution $A = a$, $B = b$, $M = m$, are the two constraints solvable? We leave it to the reader to check that they are, by setting $A' = e$, $B' = b$, $M' = m$. There is no reason to exclude this solution. It reflects a masquerading attack by e against b , which works because the message received by b is not authenticated.

3.2 Derivation Constraints and Solvability

The constraint solver does not actually use derivation rules. Instead, it has *derivation constraint transformations*, and an algorithm for how to apply them and when to stop. However, the attacker's capabilities are defined by derivation rules, and those rules are used to justify the transformations.

Here are two typical constraint transformation rules, in which T is an arbitrary list of terms, treated as an unordered set, and e is the constant attacker identity:

$$\begin{array}{ll} T \vdash u_{\text{pk}(v)} & T, u_{\text{pk}(e)} \vdash v \\ T \vdash u; T \vdash v & T, u \vdash v \end{array}$$

In each rule, if the system has a constraint matching the pattern above the line, that constraint is replaced by the constraint or constraints below the line.

The first rule is based on the attacker's ability to construct $u_{\text{pk}(v)}$ from u and v . The transformation reduces the original constraint to two simpler constraints. The second rule is based on the attacker's ability to extract u from $u_{\text{pk}(e)}$. In both cases, the logical implication is upward – ability to solve the new constraint(s) is sufficient to solve the original constraint.

The constraint transformations we have chosen are easily seen to be *sound*, in the sense that they do not introduce new solutions. Any solution to a transformed system is a solution to the original system.

The *unification* transformation is based on the trivial rule $T, u \vdash u$. It just tests for a match between the term to be derived and one that is already known. Because terms contain variables, this involves unification (which is simple syntactic unification in a free algebra). The unification transformation *eliminates* a constraint $T, u \vdash v$ when u is unifiable with v . Then the most general unifier is applied to the whole system, and retained as part of the ultimate solution.

We also need to show *completeness*: namely, that any solution to the original system is preserved by some sequence of transformations. Note that several transformations may be applicable to a given constraint system, and different choices lead to branching in the tree of transformation sequences.

The branching means that the solver performs a tree search to find a solution. It stops at the first solution. If it does not find one after searching the whole tree, it reports failure (which is often a good thing, when a solution demonstrates a compromise).

Finally, we have to show *termination*: every sequence of transformations ends in a constraint system to which no further transformations apply. Also, it must be clear whether a terminal constraint system has a solution or not. Termination is proved in [38] using a measure of the complexity of the constraint system, and showing that each transformation reduces it. In order to achieve this result, we had to restrict the order in which transformations could be applied. In particular, the sequence of constraints is significant, and the next transformation must be applied to the *first constraint that is transformable*. This restriction affects the completeness proof, of course.

A constraint of the form $T \vdash x$, where x is a variable (i.e., an uninstantiated parameter) is solvable in infinitely many ways, since an infinite number of terms can be derived from any nonempty T . Constraints like this are called *simple* and are not transformed, even by unification. A constraint system that is either empty or has only simple constraints is solved, since there is at least one successful substitution for all the remaining variables.

4 Design and Testing Choices

As simple as the constraint solver is, there are several design choices in it that have a significant effect, and such choices are made by other protocol analysis tools as well. The bounded-process limitation has already been mentioned. There are others that the reader has probably noticed that merit some discussion.

4.1 Small-System Testing

For some protocols, the bounded-process limitation is not an issue. Lowe has identified a class of protocols for which it is sufficient to analyze a “small system” consisting of one instance of each role [30]. Six assumptions about properties of the protocol are needed to yield this result. The most limiting of these is the fifth: *encrypted components are textually distinct*. This means that distinct encrypted terms in an Alice-and-Bob specification are not unifiable. Thus, a principal that receives an encrypted term knows unambiguously which message and which field it came from. (The assumption excludes trivial unifications in which a message term is simply passed on in a later message.)

The example protocol used earlier does not satisfy this condition, since the responder’s reply is unifiable with the second component of the first message. So we were just lucky to find an attack using only a small-system instance.

An example of a protocol that requires two instances of the same role is Example 1.3 in [21], rewritten in our notation:

$$\begin{aligned} A &\rightarrow B : [M_{\text{pk}(B)}, A]_{\text{pk}(B)} \\ B &\rightarrow A : [M_{\text{pk}(A)}, B]_{\text{pk}(A)} \end{aligned}$$

To compromise M , the attacker records the first message from A , and substitutes it for M in a new session between the attacker e and B . Then B returns $[[M_{\text{pk}(B)}, A]_{\text{pk}(e)}, B]_{\text{pk}(e)}$ to e . The part of the reply encrypted by $\text{pk}(B)$ is then fed back by e to a new instance of B , who re-encrypts it for e .

For this attack, the two instances of B are consecutive. Some attacks require the two instances to overlap concurrently. This kind of attack is called a parallel attack, and an example of one was given in [48]. One can construct a protocol that can be compromised *only* with a parallel attack; an example called “ffgg” was given in [35]. It can be expressed as follows, where N and N' are nonces generated by B , and M is a secret created by A . It takes two concurrent instances of the responder (B) role to compromise M .

$$\begin{aligned}
A &\rightarrow B : A \\
B &\rightarrow A : [B, N, N'] \\
A &\rightarrow B : [A, [N, N', M]_{\text{pk}(B)}] \\
B &\rightarrow A : [N, N', [N', M, N]_{\text{pk}(B)}]
\end{aligned}$$

There is a related but different question about how many different principals might be needed to implement an attack. While the number of role instances cannot be bounded in general, many role instances can share a bounded set of principals. Comon-Lundh and Cortier [14] show that it is usually sufficient to name only two principals (they use the term “agent”), one of which is the attacker. (The exception is when a protocol contains explicit tests to force principal parameters to be distinct.)

4.2 Data Types and Keys

The constraint solver does not distinguish different subsorts of the message sort. Although we talk about principals and public keys, each operation accepts any message for any of its arguments. There are two encryption operators, one for public-key encryption and the other for symmetric-key encryption.

One advantage of the constraint solver, with respect to comparable tools, is that any message can be used as a symmetric key. This permits us to model a common real-life practice of constructing keys by hashing arbitrary messages into a suitable key size. (One of Shmatikov’s major contributions was to make sure that the constraint solver’s analysis procedure was complete and terminating despite this degree of flexibility.)

Key construction is not necessarily a type violation, since an explicit hash or other type coercion operation can produce a key type from something else. Some protocol analysis approaches permit such operations, others do not.

The constraint solver can discover *type flaw* attacks, in which the attacker extracts a message field of one type – a nonce or key, for example – and re-introduces it into a message field that is supposed to be a different type, such as a principal name or an encrypted submessage. One could view the Dolev-Yao example attack above as a kind of type flaw, if the message M is expected to be a character string.

Other protocol analysis approaches impose type restrictions on operators. Dependable typing is risky to assume, however, because messages are subject to modification by attackers, who might defeat the type-tagging mechanisms by extracting data and re-tagging it. A type tag is like a concatenation $[\text{type1}, X]$ that an attacker can modify to $[\text{type2}, X]$. Cryptographic techniques can be used to protect typing, however, as shown by Heather, Lowe, and Schneider [25]. The basic idea is that an attacker cannot successfully recast $[\text{type1}, X]_K$ as type2 without knowing K .

4.3 Concatenation

The concatenation operator $[X, Y]$ is often modeled as a non-associative binary operator, but it may also be treated as associative or n -ary. The choice can make a difference. Consider the protocol:

$$\begin{aligned}
A &\rightarrow B : [A, N_a]_{\text{pk}(B)} \\
B &\rightarrow A : [N_a, [N_b, B]]_{\text{pk}(A)} \\
A &\rightarrow B : (N_a)_{\text{pk}(B)}
\end{aligned}$$

This is a slight variation (field order in the first message) on Lowe’s modification of Needham-Schroeder [29]. If an attacker X initiates a session with B , masquerading as A and using X in place of N_a , X receives a response encrypted for A :

$$\begin{aligned}
X/A &\rightarrow B : [A, X]_{\text{pk}(B)} \\
B &\rightarrow X/A : [X, [N_b, B]]_{\text{pk}(A)}
\end{aligned}$$

where X/A is X masquerading as A . But, then X can initiate a session in which A is the responder:

$$\begin{aligned}
X &\rightarrow A : [X, [N_b, B]]_{\text{pk}(A)} \\
A &\rightarrow X : [[N_b, B], [N_a, A]]_{\text{pk}(X)}
\end{aligned}$$

After this, X knows B ’s nonce N_b , and can finish the first session with B , while still pretending to be A . This example was discovered using the constraint solver.

This could be viewed as a type flaw, since A is asked to accept $[N_b, B]$ as a nonce. But the vulnerability also hinges on recognizing $[X, [N_b, B]]$ as a binary concatenation. If the protocol had been specified to use a ternary concatenation $[N_a, N_b, B]$ in the second message, the attack fails, because A is expecting a binary concatenation in the first message of the responder role.

4.4 Free Algebra

The constraint solver uses a free message term algebra, following the example of the strand space formulation [46]. In a free algebra, two ground terms can be equal only if they are identical. This brings some conveniences but also some limitations. The completeness and termination proofs for the constraint solver depend on this property of the message algebra.

One limitation is that if we have an encryption operator, there is no explicit decryption operator. So, in particular, the original Dolev-Yao algebra is not free. At first sight, this might not appear to be limiting in practice, because, first, typical real protocols do not make explicit use of a decryption operator, and, second, because the ability of honest principals and the attacker to decrypt messages can be expressed implicitly. For honest principals, a role might simply receive $M_{\text{pk}(A)}$ and place M in a return message. An attacker can use a derivation rule like $u_{\text{pk}(e)} \vdash u$.

Yet, there is a limitation, because it prevents the attacker from decrypting a message that has never been encrypted. We can give a simple, albeit artificial, example of why this matters. Consider the following protocol, shown in Figure 2. This is clearly an unrealistic protocol – one could not even express it in the Alice-and-Bob form – but our protocol model does not exclude it. In the figure, S_K is the symmetric encryption of S with the key K . Symmetric decryption is not required to specify the protocol, but it would be written S_K .

In this protocol, it is assumed that S is a secret, K is a symmetric key that is shared by certain trusted principals but not the attacker, C is a symmetric key

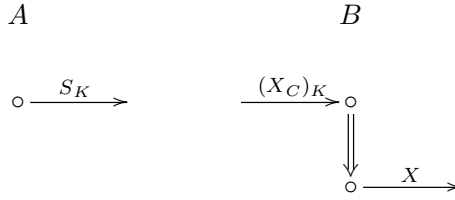


Fig. 2. Compromise Hidden By Free Algebra

known to everyone, and X is just a parameter representing an unknown message.

When the initiator sends S_K , an honest responder can interpret this as $((S_{\bar{C}})_C)_K$, which is acceptable with $X = S_{\bar{C}}$. The responder then transmits X , and the attacker encrypts it with C to get $X_C = (S_{\bar{C}})_C = S$. This compromise cannot be found if the message algebra is free, since the decryption operator is not available, so $X = S_{\bar{C}}$ is not a permissible substitution.

This example was given in [36], where it is shown that symmetric decryption does not introduce extra vulnerabilities if all encryption is received with context. In other words, if every encrypted term has some structure (i.e., it is not just a variable), there are no vulnerabilities that are hidden from analysis using a free algebra. This condition was called *EV-freedom*. The result in [36] was proved only for symmetric encryption. It was extended to public-key encryption (with *PEV-freedom*) by Lynch and Meadows [32].

5 Concluding Remarks

The constraint solver employs term rewriting in a unique way. There are other approaches to modeling and analyzing protocols that use term rewriting more directly. We have already mentioned the Dolev-Yao model of ping-pong protocols, but more general models were needed to handle realistic protocols. Early tools like the Interrogator [37] and the NRL protocol analyzer [34] had message algebras with both symmetric and asymmetric encryption, rules for state transitions of role instances, and rules for updating the set of terms known to the attacker. The NRL analyzer went further by identifying *narrowing* as the procedure for determining when a state transition could generate a given message term that is a knowledge goal for the attacker. Narrowing is essentially unification in an equational theory, as needed to represent cancellation of encryption and decryption, and the NRL analyzer implemented an existing narrowing algorithm.

The MSR (multiset rewriting) notation is a formal meta-notation motivated by a linear logic concept [12]. The multiset is a global bag of facts about the current state of each role instance and the attacker knowledge. Additions to attacker knowledge and replacement of role instance states are both easily represented as term rewriting rules. Linear logic offers a special formalism for uniquely originating nonces, using an existential quantifier that is assumed to generate new symbols.

For example, the rule:

$$B_0(), N_1(x) \rightarrow \exists y. B_1(x, y), N_2(x, y)$$

is applicable when the multiset contains at least one instance of the “facts” $B_0()$ and $N_1(x)$, and replaces them by the pair of facts on the right, after creating a new nonce symbol y . A fact $B_i(\dots)$ is an instance of the “ B ” role in state i with a given list of parameter data (which is empty in state 0). A fact $N_1(x)$ means that the data term x has been transmitted and is available in the network to be received. The facts on the right indicate that the pair (x, y) is to be transmitted, and B transitions to state 1 with values for parameters x and y .

This model was implemented in a tool designed to study linear logic. A special tool is not really necessary to simulate this model; existing applications of model checking tools use tricks such as sequence numbering to simulate nonces.

Current frontiers in protocol analysis are in three main directions. One is to support a wider choice of encryption operators. Much work has been done already to handle exclusive-or encryption [47] and Diffie-Hellman key distribution, based on modular exponentiation [31,24]. These operators have a commutative character that is challenging to handle algebraically in an efficient way. Two of the more recently considered unusual operators are bilinear pairings [33], as used in elliptic curve cryptography, and zero-knowledge proofs [2].

The second frontier is to create tools that support computational models in some way. Work in this category includes [11,7]. Finally, the third frontier is to broaden the scope of security goals that can be analyzed. Some of these topics are voting, anonymity, contract-signing, trust management, and group protocols with evolving membership. The field has gotten considerably broader and deeper in the last few years!

References

- [1] M. Abadi and P. Rogaway. Reconciling two views of cryptography. *J. Cryptology*, 15(2):103–127, 2002.
- [2] M. Backes, M. Maffei, and D. Unruh. Zero-knowledge in the applied pi-calculus and automated verification of the direct anonymous attestation protocol. In *Symposium on Security and Privacy*, pages 202–215. IEEE Computer Society, 2008.
- [3] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *ACM Conference on Computer and Communications Security*. ACM SIGSAC, 2003.
- [4] D. Basin, S. Moedersheim, and L. Vigano. Constraint differentiation: A new reduction technique for constraint-based analysis of security protocols. In *ACM Conference on Computer and Communication Security*. ACM SIGSAC, 2003.
- [5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO '98*, number 1462 in LNCS, pages 26–45. Springer, 1998.
- [6] B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. *14th IEEE Computer Security Foundations Workshop*, IEEE Computer Society, 2001, 82–96.
- [7] B. Blanchet. A computationally sound mechanized prover for security protocols. In *Symposium on Security and Privacy*, pages 140–154. IEEE Computer Society, 2006.
- [8] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology - CRYPTO '98*, volume LNCS 1462, pages 1–12. Springer, 1998.
- [9] D. Bolognani. Towards the formal verification of electronic commerce protocols. In *IEEE Computer Security Foundations Workshop*, pages 133–146. IEEE Computer Society, 1997.
- [10] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

- [11] R. Canetti and J. Herzog. Universally composable symbolic analysis of cryptographic protocols. Technical Report 334, IACR Cryptology ePrint Archive, 2004.
- [12] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *12th IEEE Computer Security Foundations Workshop*, pages 55–69. IEEE Computer Society, 1999.
- [13] E. Cohen. Taps: A first-order verifier for cryptographic protocols. In *Computer Aided Verification*, volume LNCS 1855, pages 568–571. Springer, 2000.
- [14] Hubert Comon-Lundh and Véronique Cortier. Security properties: two agents are sufficient. Research Report LSV-02-10, ENS de Cachan, 2002.
- [15] R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *9th Int. Static Analysis Symp. (SAS)*, volume LNCS 2477, pages 326–341. Springer-Verlag, 2002.
- [16] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In *Formal Methods and Security Protocols*, 1998. LICS '98 Workshop.
- [17] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8), August 1981.
- [18] S. Doghmi, J. Guttman, and F.J. Thayer. Completeness of the authentication tests. *ESORICS '07*, 2007.
- [19] S. Doghmi, J. Guttman, and F.J. Thayer. Searching for shapes in cryptographic protocol. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS)*, LNCS 4424, 2007.
- [20] D. Dolev, S. Even, and R. Karp. On the security of ping-pong protocols. *Information and Control*, 55:57–68, 1982.
- [21] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29:198–208, 1983. Also STAN-CS-81-854, May 1981, Stanford U.
- [22] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Formal Methods and Security Protocols*, Federated Logic Conference, 1999.
- [23] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *24th IEEE Symposium on Foundations of Computer Science*, 1983.
- [24] S. Escobar, C. Meadows, and J. Meseguer. A rewriting-based inference system for the NRL Protocol analyzer and its meta-logical properties. *Theoretical Computer Science*, 367(1), 2006, 162–202.
- [25] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *13th IEEE Computer Security Foundations Workshop*, pages 255–268. IEEE Computer Society, 2000.
- [26] N. Heintze and J. Tygar. A model for secure protocols and their compositions. *IEEE Transactions on Software Engineering*, 22(1):16–30, January 1996.
- [27] A. Huima. Efficient infinite-state analysis of security protocols. In *Workshop on Formal Methods and Security Protocols*. FLOC, 1999.
- [28] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *5th ACM Conference on Computer and Communications Security*, pages 112–121. ACM SIGSAC, 1998.
- [29] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer-Verlag, 1996.
- [30] G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2/3):89–146, 1999.
- [31] C. Lynch and C. Meadows. Sound approximations to Diffie-Hellman using rewrite rules. LNCS 3269, 2004.
- [32] C. Lynch and C. Meadows. On the relative soundness of the free algebra model for public key encryption. *ENTCS*, 125(1):43–54, 2005.
- [33] L. Mazare. Computationally sound analysis of protocols using bilinear pairings. In *WITS 2007*, pages 6–21. IFIP, 2007.
- [34] C. Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–36, 1992.
- [35] J. Millen. A necessarily parallel attack. In *FLoC Workshop on Formal Methods and Security Protocols*, 1999.

- [36] J. Millen. On the freedom of decryption. *Information Processing Letters*, 86(6):329–333, June 2003.
- [37] J. Millen, S. Clark, and S. Freedman. The Interrogator: protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.
- [38] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *8th ACM Conference on Computer and Communication Security*, pages 166–175. ACM SIGSAC, November 2001.
- [39] J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using Mur ϕ . In *IEEE Symposium on Security and Privacy*, pages 141–154. IEEE Computer Society, 1997.
- [40] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–998, December 1978.
- [41] L. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1):85–128, 1998.
- [42] A. W. Roscoe. Modelling and verifying key-exchange protocols using CSP and FDR. In *8th IEEE Computer Security Foundations Workshop*, pages 98–107. IEEE Computer Society, 1995.
- [43] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *14th IEEE Computer Security Foundations Workshop*, pages 174–190. IEEE Computer Society, 2001.
- [44] S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9):741–758, September 1998.
- [45] D. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1), 2001, 47–74.
- [46] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.
- [47] M. Tuengerthal, R. Küsters, and M. Turuani. Implementing a unification algorithm for protocol analysis with XOR. Research report, Christian-Albrechts-Universität zu Kiel, 2006.
- [48] T. Woo and S. Lam. A lesson on authentication protocol design. *ACM Operating Systems Review*, pages 24–37, 1994.